Error Handling

Introduction

In the Play+ ecosystem, we believe that errors are opportunities to build trust, not failures to hide. This helper is based on the concept of Resilience by Design , where error handling transforms from a reactive, defensive practice into a proactive, trust-building experience.

The playerror helper provides a comprehensive, user-centric approach to managing errors throughout your application. It automatically enriches errors with context, classifies them by severity, and provides clear recovery paths. This aligns directly with our Intuitive pillar by making error handling predictable and user-friendly, our Adaptive pillar by gracefully handling different error scenarios, and our Engaging pillar by maintaining user trust even when things go wrong.

Package Info

The Play+ error handling helper and its associated configurations are included by default in the Golden Path starter kit.

Package / Path	Golden Path (Recommended)
Pre-installed	/system/play.error.ts
Uplift Path	npm install @playplus/error

Folder Reference

The error handling helper and its configuration follow our standardized folder structure for core system logic.

File / Directory	Purpose & Guidelines
system/play.error.ts	The core error handling service. It provides utilities for error reporting, classification, and user feedback.

File / Directory	Purpose & Guidelines
config/play.error.config.json	User-overridable configuration for error messages, PII redaction, and handling strategies.
src/app/core/global-error-handler.ts	Angular global error handler that integrates with the Play+ error system.
src/app/services/playerror.service.ts	Angular service wrapper that provides component-specific error handling utilities.
src/app/components/error-boundary/	Error boundary components for React-style error isolation.

Helper - Pillars Alignment

The playerror helper is a direct implementation of our core design pillars, focused on building user trust through intelligent error handling.

Pillar	How This Helper Aligns
Intuitive	Primary Pillar: Provides clear, actionable error messages and predictable recovery paths that users can easily understand and follow.
Adaptive	Gracefully handles different error scenarios (network, validation, server) with appropriate responses and fallback strategies.
Engaging	Maintains user engagement by preserving work when possible and providing helpful recovery options instead of frustrating dead ends.

Helper Overview

The playerror toolchain is a comprehensive error management system that automates the complex aspects of error handling while providing developers with simple, powerful tools.

Instead of manually writing try-catch blocks and error reporting logic, developers can rely on the Play+ system to handle errors intelligently.

It automates error quality control in several key ways:

- • Automatic Error Capture : Global handlers catch uncaught exceptions and route them through the Play+ system
- • Contextual Intelligence : Automatically enriches errors with user, session, and application context
- • Smart Classification : Determines error type and severity to route to appropriate handling strategies
- • User-Centric Feedback : Transforms technical errors into clear, actionable user messages
- • Recovery Paths : Provides clear next steps for every error scenario

The goal is to make robust error handling the default path, not an afterthought.

Config Options

Global error handling settings are managed in config/play.error.config.json . These values control error messages, PII redaction, and handling strategies.

Config Key Table

Config Key	Default Value	Description	Recommended Value
defaultMessages.server	"We're fixing it!"	Default message for server errors.	Custom message for your app
defaultMessages.notFound	"Item not found."	Default message for 404 errors.	Custom message for your app
defaultMessages.auth	"Please log in again."	Default message for authentication errors.	Custom message for your app
defaultMessages.forbidden	"You lack permission."	Default message for 403 errors.	Custom message for your app

Config Key	Default Value	Description	Recommended Value
defaultMessages.timeout	"Connection timed out."	Default message for timeout errors.	Custom message for your app
defaultMessages.network	"Check your internet connection."	Default message for network errors.	Custom message for your app
defaultMessages.rateLimit	"Too many requests."	Default message for rate limiting errors.	Custom message for your app
defaultMessages.validation	"Please check your input and try again."	Default message for validation errors.	Custom message for your app
defaultMessages.unknown	"Something went wrong. Please try again."	Default message for unknown errors.	Custom message for your app
piiFields	["password", "email", "token", "apiKey", "secret", "authorization", "cookie", "sessionId"]	Fields to redact from error logs.	Add any additional sensitive fields
enableAutoReporting	true	Whether to automatically report errors to monitoring systems.	true
enableUserFeedback	true	Whether to show user-friendly error messages.	true
enableRecoveryOptions	true	Whether to provide recovery options for errors.	true

Helper Methods

Core Methods

Method Name	Description	Signature
report	Reports non-critical errors with context and user feedback.	report(error: Error, context?: ErrorContext): void
reportCritical	Reports critical errors with immediate user notification and recovery options.	reportCritical(error: Error, context?: ErrorContext): void
showToast	Shows a toast notification with error message.	showToast(message: string, type: 'error' | 'warn' | 'info'): void
showModal	Shows a modal dialog with error details and recovery options.	showModal(title: string, message: string, actions: ErrorAction[]): void
showInline	Shows inline validation error for form fields.	showInline(elementId: string, message: string, type: 'error' | 'warn'): void
withContext	Creates a scoped error reporter with predefined context.	withContext(context: ErrorContext): ErrorReporter

Angular Integration

PlayErrorService

Angular service wrapper that integrates with Play+ logging and provides component-specific error handling utilities.

Global Error Handler

Automatically catches and processes uncaught exceptions throughout the application.

Error Boundary Component

Provides React-style error isolation for complex components.

Usage Examples

React: Basic Error Reporting

This example shows how to report errors with context and provide user feedback.

Angular: API Error Handling

This example shows how to handle API errors with retry functionality and user feedback.

Basic Usage Examples

Additional Info

Why We Created This Helper

Error handling is critical for user experience, but implementing it correctly is complex and error-prone. Without a dedicated system, developers would need to:

- Manually write try-catch blocks for every async operation
- Create custom error reporting logic for each component
- Implement user feedback mechanisms for different error types
- Handle PII redaction and security concerns manually
- Build recovery paths and retry mechanisms from scratch

This leads to inconsistent error handling and poor user experiences. The playerror helper automates these common patterns and provides a simple, consistent API that makes robust error handling the default path.

Error Classification

The system automatically classifies errors into three categories:

System Errors (Critical)

- Network failures : Connection timeouts, DNS failures
- Server errors : 5xx responses, service unavailable
- Runtime errors : JavaScript exceptions, memory issues

Handling : Immediate reporting, user modal, retry options

User Errors (Non-Critical)

- Validation errors : Invalid input, missing required fields
- Business rule violations : Insufficient permissions, quota exceeded
- Authentication errors : Expired tokens, invalid credentials

Handling : Toast notifications, inline feedback, guided correction

Environmental Errors (Adaptive)

- Offline state : No internet connection
- Slow connections : Timeout warnings
- Browser limitations : Storage blocked, feature unsupported

Handling : Adaptive UI, cached data, transparent communication

Best Practices

DO

- Use contextual reporting : Always provide component and action context
- Classify errors appropriately : Use report() for non-critical, reportCritical() for critical
- Provide recovery paths : Every error should have a clear next step
- Test error scenarios : Simulate network failures, invalid inputs, server errors
- Monitor error trends : Track error rates, recovery rates, and user impact
- Use error boundaries : Wrap complex components to prevent cascading failures

DON'T

- Show technical errors to users : Never expose stack traces or technical details
- Throw errors without reporting : Always use playerror.report() or reportCritical()
- Ignore user context : Don't report errors without user and session information
- Use generic error messages : Provide specific, actionable feedback
- Forget recovery options : Every error should have a way forward
- Log sensitive data : PII is automatically redacted, but be mindful of additional context

Security Considerations

- PII Protection : Sensitive data is automatically redacted from logs

- Error Sanitization : Error messages are sanitized before display
- Secure Reporting : Error reports are sent over secure channels
- Access Control : Error logs are protected by appropriate access controls

Forbidden Patterns

Console Error Logging

Generic Error Messages

Technical Error Exposure

Missing Recovery Options

Required Patterns

Contextual Error Reporting

Error Classification

User Feedback

Recovery Paths

Testing

Error Simulation

Error Monitoring

Monitoring and Analytics

Error Metrics

- Error Rate : Percentage of requests that result in errors
- Recovery Rate : Percentage of errors that users successfully recover from
- Mean Time to Recovery (MTTR) : Average time to resolve errors
- User Impact : Number of users affected by errors

- Error Distribution : Types and frequencies of different errors

Alerting

Integration with Other Play+ Systems

Logging Integration

Security Integration

Performance Integration

Compliance and Monitoring

GDPR Compliance

- Data Minimization : Only collect necessary error context
- Right to Erasure : Error logs can be purged on user request
- Transparency : Users are informed about error reporting
- Security : Error data is encrypted in transit and at rest

Configuration Example

Migration Guide

From Console Errors

From Try-Catch Blocks

From Global Error Handlers

Developer Checklist

- Do I use playerror.report() or reportCritical() for all error handling?
- Do I provide component and action context with every error report?
- Do I show user-friendly error messages instead of technical details?
- Do I provide recovery options for every error scenario?
- Do I test error scenarios including network failures and invalid inputs?
- Do I use error boundaries for complex components?

- Do I monitor error rates and recovery rates?
- Do I avoid logging sensitive data in error context?

Summary

The Play+ error handling system provides:

- Proactive Error Management : Catch and handle errors before they reach users
- User-Centric Feedback : Clear, actionable error messages
- Contextual Intelligence : Rich error context for debugging
- Security by Default : Automatic PII redaction and sanitization
- Recovery Paths : Every error has a clear way forward
- Monitoring Ready : Built-in metrics and alerting
- Compliance Ready : GDPR and security compliant
- Developer Friendly : Simple API that enforces best practices

90% of error handling is automated. Focus on the 10% that matters.